The Commandments of COTS:
Still in Search of the Promised Land

David J. Carney, Patricia A. Oberndorf
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213
{djc, po}@sei.cmu.edu

1. Introduction

Within the past few years, organizations that acquire software-intensive systems have undergone a remarkable shift in emphasis toward use of existing commercial products. This shift is especially noticeable in the U.S. Government procurements, particularly those of the Department of Defense (DoD). Many current Requests for Proposals (RFPs) being issued by the Government now include a mandate concerning the amount of "COTS" (commercial off-the-shelf) products that must be included. Supporting this are statements from senior officials that indicate a policy for using COTS products as much as possible is intended to be Department- or even Government-wide.

This interest in COTS products requires examination both in terms of its causes and effects, and also in terms of its benefits and liabilities. In this paper we offer some observations on all of these, and voice some specific concerns and criticisms. We stress that our observations are essentially cautionary, not condemnatory: it is obvious to almost any observer that the huge growth in software costs will continue, not abate, and that appropriate use of commercially-available products is one of the remedies that might enable us to acquire needed capabilities in a cost-effective manner. Where use of an existing component is both possible and feasible, it is no longer acceptable for the Government to specify, build, and maintain a comparable product; clearly an existing commercial solution is called for.

But on the other hand, we also notice many disturbing signals abroad in the land. Many statements, whether high-level policies or otherwise, suggest a reluctance to admit that as with anything else, there can be a nontrivial downside to using COTS products. Like any solution to any problem, there are drawbacks as well as benefits: while it may not be readily apparent to all observers as yet, many unhappy trade-offs exist when embracing a commercial basis for the Government's software systems.

The critical point is that using COTS components in any given circumstance might help, but is not guaranteed to, and such use may even cause greater problems. Acquisition managers must understand that use of a COTS component may be a reasonable solution, but that its use should be the product of analysis, reasoning, and engineering decisions, not the result of jumping on the latest bandwagon. Hence we express, using a somewhat whimsical structuring device, some specific items to consider when planning to use COTS products. Unlike the Commandments of the Exodus, our commandments are not graven in stone. But they are surely needed: just as was true in that far-off time, we see abundant evidence that people are still willing to worship before false idols.

2. The Commandments

We suggest ten "commandments" to consider. We state them briefly here, and expand on each later in this paper

I. One more time: Do not believe in Silver Bullets

"COTS is the answer" is the latest in a sequence of Silver Bullet slogans, each of which was expected to produce an immediate and painless solution to a perceived technology crisis. Brooks really was right: Silver Bullets don't work, and often make things worse.

II. Use the term precisely (and demand like behavior from others)

The term "COTS" is currently a widely-used buzzword with many possible interpretations. In spite of a precise definition in the Federal Acquisition Regulations, it is still possible for significant misunderstandings to arise between Government and vendors.

III. Understand the impact of COTS products on the requirements and selection process

Use of COTS products has significant impact on both the specification of requirements and the evaluation of proposals, an impact that must be understood early in the acquisition process.

IV. Understand their impact on the integration process

Integration of COTS components is no simpler than integration of proprietary components, and may in fact be more difficult.

V. Understand their impact on the testing process

Testing and validation of COTS-based systems is a radically different process than testing and validating proprietary systems

VI. Realize that a COTS approach makes a system dependent on the COTS vendors

Vendor support for the commercial components in a COTS-based systems is critical to the success of that system; many unforeseen problems can accompany a commercial system after deployment.

VII. Realize that maintenance is not free

The presence of COTS components does not necessarily mean low maintenance costs; on the contrary, their presence can cause complex problems in system upgrade and system maintenance. These problems may well exceed the maintenance cost of a home-grown system.

VIII. You are not absolved of the need to engineer the system well

A system that is composed of components from diverse sources and suited to your particular needs will not come together by itself.

IX. Just "doing COTS" is not an automatic cost-saver

It may be observed that the availability of off-the-shelf parts has brought down the costs in other industries; but in software development, there are offsetting costs to consider and manage.

X. Just "doing COTS" must be part of a large-scale paradigm shift

A change in mindset is as important as any change in technology when composing systems from parts created by others (as opposed to designing and building everything from scratch).

I. One more time: Do not believe in Silver Bullets.

There seems to be universal agreement throughout the software community that "there is no silver bullet," and that the lessons found in Brooks' famous paper[1] have been learned. Yet the silver bullet mentality seems to reappear in the Government with remarkable frequency. We note a long series of initiatives (Ada, CASE tools, SEEs, CIM, and reuse being the most prominent) that placed all-encompassing hopes on particular technologies to answer impending crises. The results seldom matched the expectations. Nor were these technologies in themselves necessarily failures (e.g., Ada's inability to find broad success was not, we feel, due to any inherent technical weakness in the language itself; many CASE tools are indeed valuable and useful products).

Instead, we suggest that the evidence shows that the complex technological problems found in software-intensive systems demand complex solutions, not simple ones. While the desire for a simple and uncluttered remedy is understandable, we suggest that this desire is unlikely to be satisfied. Thus, we assert, it is the single-mindedness of the response, not whatever technology is the silver bullet at hand, that is inherently flawed. A technological crisis, including all of the "software crises" that have come and gone, is generally the product of many factors. While there may be one overriding factor at the base of the problem, there are likely to be many other contributing factors that participate and catalyze; serious technological difficulties tend to result from the interaction of factors rather than from any single one alone. Thus, an approach that chooses a single factor, isolates it, and then hopes for a universally beneficial outcome will usually have little success.

We perceive that the current movement toward COTS products is rapidly taking on such a tone. The growth in the cost and complexity of software systems is a difficult problem, but to focus only on a commercially-based solution is to misunderstand the problem's difficulty. To mandate *a priori* that some arbitrary percentage of any system should be COTS products will almost certainly not have the desired effect. Instead, the real need is for:

- thoughtful policies about the types of systems that will or will not benefit from a commercial approach,
- guidelines about the hard trade-offs made when incorporating COTS products into systems,
- handbooks describing recommended processes and procedures about integrating multiple commercial products,
- upgrade strategies covering multiple vendors,
- recommendations about when *not* to use a commercial approach.

Further, these items must be recognized as only part of the solution, and must be accompanied by similar considerations of the complementary problems -- system distribution, interface standards, legacy system reengineering, and so forth -- with which a COTS-based approach must be integrated.

In short, it is necessary that using COTS products be seen as one potential strategy in a complex solution space, no more and no less. A simplistic mandate to "use COTS as much as you can" solves little and, as will be seen below, may well make matters much worse.

---

1. Brooks, Jr., F. P. "No Silver Bullet - Essence and Accidents of Software Engineering," Computer, 20(4), April 1987, 10-19

II. Use the term precisely (and demand like behavior from others)

The U.S. Senate Bill 1597 provides a definition of both "commercial item" and "nondevelopmental item." The thrust of these definitions is to clarify the Government's understanding of what it means to be a "commercial product."[2] While these definitions are useful, they leave some issues unresolved. For instance, they specify that a product must be either "sold, leased, or licensed to the general public, offered for sale, lease, or license to the general public or be made available ...within a reasonable period." But what is "a reasonable period"? And in a different vein, is there any sense in which the product must have been advertised? These questions are not unimportant, since they can have major importance when evaluating proposals for source selection.

A further ambiguity stems from the unfortunate parallelism with the terms "GOTS" (Government off-the-shelf) and "MOTS"(Modified off-the-shelf). Both are thought of as related to, and sometimes even considered subsets of COTS. In the former case, a GOTS component is one developed by and owned by the Government (generally including the source code). In the latter case, a MOTS component is one in which some alteration has been made to an existing component or product for the purpose of a specific acquisition. However, to consider all of these as roughly synonymous is dangerous. For instance, given that one prime motivation for choosing COTS is to lower maintenance costs, then the term should apply only when source code is unavailable and maintenance impossible (except by its vendor). In that sense, "MOTS" is fundamentally different from COTS, since any code that is modified in any way for a specific acquisition will of necessity need ongoing maintenance for the life of the acquired system.

Aside from the precise nuance of meaning, there is also the issue of scope: what size of component do we include when we say "COTS"? Is it meant to be all-inclusive, covering any software component from small CASE tools to an entire Air Traffic Control system?[3]

What drives the Government's current bias toward COTS products is the intuitive belief that in constructing a complex system, there are often some cases where "the item I want already exists on someone else's shelf." In this simplistic sense, it is moot whether that shelf is commercial or otherwise, and thus there is a significant commonality between a COTS component and a nondevelopmental item (NDI), since in both cases the acquiring organization desires to delegate the creation of certain functionality elsewhere. Whether the source of that functionality is the commercial world or merely somewhere else in Government is unimportant. In short, the key issue is the Government's heightened willingness to consider the classical "make vs. buy" question often faced by industry.

But in any sense other than this simple one, there are shades and nuances of meaning that may or may not be significant for a particular acquisition: these subtly different meanings provide a dangerous level of ambiguity. A precise specification of the intended meaning of the term "COTS" is manifestly necessary for each acquisition.

---

2. The Federal Acquisition Regulations (FARS) are currently being amended to reflect the Senate definitions.
3. If so, then a single technical policy is being applied equally and across the board to systems of ten thousand lines and systems of ten million lines; in short, "one size fits all."

III. Understand the impact of COTS products on the requirements and selection process

There is widespread agreement throughout the software community on the importance of a requirements specification; anecdotal evidence suggests that the requirements specification can be the single most important factor in the success of an acquisition. There are, therefore, some obvious impacts that a COTS bias will have both on the specification of requirements and on the evaluation of proposals.

A bias toward COTS software products necessarily implies either that some software requirements will be written to describe existing products, or that the requirements are malleable enough to be implemented with a variety of existing products. Said differently, someone (whether the requirements author or not) must choose which requirements can bend to the exigencies of the marketplace and which cannot. Nor is this unrealistic: requirements are not, after all, written in a vacuum, and for any given component of a software-intensive system, the author of its requirements can distinguish absolute from desirable properties of the system. But for those requirements that *are* expected to be amenable to a commercial solution, then the requirements author must have some notion of how bidders will respond. For instance, if an acquisition of some CASE tools includes a project management tool, and the expectation is for a COTS item from most bidders, then the author of its requirements thus must have adequate knowledge of the existing CASE marketplace, which will then guide the description of required functional features. Anything else would be self-contradictory: soliciting bids for a commercial product, yet describing functional capabilities for which no commercial instances exist. Paradoxically, the requirements must also be sufficiently generic, since unless we assume that the author of the requirements has predecided on a specific product, the requirements must be broad enough to accommodate the differences between comparable commercial products.

A COTS bias can also have an impact on the evaluation process. Selection criteria become difficult to define when choosing between two comparable products such as FrameMaker or Interleaf: given these choices, what clearly distinguishes them? What aspects of selecting one or the other will survive a challenge to an award? Another difficulty for source selection stems from the looseness of the term “COTS” described earlier: will each acquisition refine what it means by “COTS”? Will all acquisitions abide by the Government-wide definition (e.g., the FARs)? Are there some minimal yearly sales figures by which some component is *acceptably* COTS? Or might it be enough that the bidder proposes some arbitrary component and states that it is “commercially available”? Even worse, there have even been suggestions (at least in the presence of the authors) that the COTS directives might eventually be phrased such that “at least 50% of the bidder’s proposal must be COTS.” If so, how is the presence of “50%” determined? Half of all CSCIs? Half the lines of code (which cannot be determined with COTS products)? Half the “shall” requirements? Presuming that the 50% figure *could* be determined, does it also logically follow that with all other things being equal, that 65% COTS is de facto preferable to 55%?

IV. Understand the impact of COTS products on the integration process.

The current thrust toward COTS products takes place in the context of “COTS-based systems.” That is, while there are demonstrable benefits in preferring to purchase a standalone Oracle DBMS rather than creating its functional equivalent from scratch, the current COTS initiatives are really focused on systems: complex groupings of components, interacting in diverse ways, and in

which introducing commercial components will simultaneously result in lower costs (e.g., due to lowered maintenance costs for the Government) as well as giving the system a “plug and play” character (e.g., because of the expected high degree of integrability).

The phrase “plug and play” is significant, and is the unspoken motivator for much of the current interest in COTS products. It conjures up a software environment wherein heterogeneous components can be easily inserted or replaced, and in which components interconnect and interoperate with ease. It is based on the conditions found in the hardware world, where some degree of “plug and play” really does exist; boards from one maker, cables from another, printers, monitors, and keyboards can all be purchased, replaced, and upgraded independently and easily.

In terms of software, this tempting picture is a myth, and will remain a myth for several years. Software as typically written today shares few of the integrability characteristics of hardware. There are no widely-observed functional boundaries of a component (i.e., while a keyboard performs a single bounded role in the hardware world, there is no such analog for a design tool). Even for those software items that do have a relatively clean demarcation of functionality, such as a database management system, the vendors of such products tend to bundle that functionality with other capabilities (e.g., a DBMS might provide a GUI builder, 4GL processors, etc.). Thus, individual products are actually unique collections of tools and utilities whose internal interfaces are generally obscured. These products are often extremely useful, but are not easily “plugged,” since different products exhibit or obscure different interfaces, and a fatal mismatch is almost certain when one product is replaced by another.

The “plug and play” notion also rests on assumptions about data that different tools will share, again by analogy to the hardware world. Once again, it is necessary to note the nontrivial differences between the two worlds. For hardware, low-level protocols tend to be semantically simple (as are agreements on physical things like pin configurations). By contrast, the information that crosses software interface boundaries is semantically far richer. A data “item” for a graphical design tool will include many levels of information unique to that tool; its meaning may also incorporate details of access control, relationships with other data items, ownership, longevity, and so forth. Vendors of such components tend to keep details like these private (which is a logical outcome of the decision to incorporate multiple functionalities into a single product). The result is that until there are wide agreements from many software vendors about shared data, then even if it is possible to “plug” a new component into a software system, it is not very likely that one will be able to “play.”

V. Understand the impact of COTS products on the testing process.

It is a truism that testing a system is quite different from testing its constituent parts. This difference is especially pronounced when the system makes use of previously-existing software components. This fact is dramatically demonstrated in the findings of the investigation of the recent explosion of the European Space Agency’s Ariane-5 rocket, where software from the earlier version of the rocket (Ariane-4) was reused:

> The error occurred in ... a software module [that] computed meaningful results only before lift-off. As soon as the launcher lifts off, this function serves no purpose. [The purpose of this function] is based on a requirement of Ariane-4...
>
> ***The same requirement does not apply to Ariane 5, which has a different preparation***

> ***sequence and it was maintained for commonality reasons, presumably based on the view that, unless proven necessary, it was not wise to make changes in software which worked well on Ariane 4***.[4] (emphasis ours)

We cite this instance because it provides a highly relevant lesson when deciding to use any previously-written component (which certainly includes COTS components) in a system.What types of testing, both at the unit level and at the system level, are possible? And what types of testing are necessary? In the example cited above, there was a requirement that drove the creation of a module for the earlier system. The module was reused later, but the requirement overlooked (or, more likely, forgotten), and thus whatever testing was performed made no provision for it. So when a system designer today is faced with the choice of using a COTS component, how is he to determine the testing that will be necessary? What requirements drove the creation of that component? Are they documented? Are they complete? Even at the sub-system level, how does one do unit testing for a COTS component?

Current research is only beginning to grapple with these questions; it is vital that we have some better notions of how they might be answered. Widespread incorporation of COTS components into systems may not result in failures as spectacular or as costly as that of Ariane-5. But for at least some systems, similar misunderstanding about requirements will occur, and failures they will most certainly be.

VI. Realize that a COTS approach makes a system dependent on the COTS vendors

The role of the components' vendors can be a decisive factor in successfully creating and maintaining a commercially-based software system. There are several aspects to this, of which we suggest three as especially significant.

First, what standards govern the product in question? In spite of the current interest in the role of standards for commercial products, there is still the danger of a system becoming overly dependent on a particular vendor's product. This commonly occurs when a product has a number of desirable but non-standard features. Such features are often attractive, but they exact a price, since a system that relies on them will result in what is commonly called "vendor lock."

Second, does the vendor supply adequate documentation for the component in question? Some products offer extensive and useful documentation, but this is by no means normative. (This point can be easily verified by any system administrator who has tried to follow installation instructions.) Nor is installation documentation sufficient: do programmer reference manuals or maintenance manuals exist? More to the point: are they well-written? In any case, are they accurate? Nor is documentation the only issue: what kind of user support is available? Is the company responsive to user inquiries? If the component is to be used in a critical system, what is the availability of that support (e.g., is it twenty-four hour support)?

Third, if the component is to be part of a system that will operate for several years, what are the probabilities that the company will exist for that time? There are many anecdotes throughout the software community about products' vendors going out of business leaving clients with nonfunctional tools and unretrievable data: such anecdotes are not entirely legendary. Finally, even assum-

---

4. "ARIANE 5:Flight 501 Failure." Report by the Inquiry Board, Prof. J.L. Lyons, Chairman, Paris, 19 July, 1996.

ing that the company exists, how long will it support the COTS product in question? Vendors often phase out their support for any given product: would such an occurrence have an impact on the maintenance of the system in question?

These questions are neither new nor profound; to a large extent, they have been relevant in many acquisitions for years (e.g., questions like these are always in the mind of a buyer of CASE tools). But they demand reasonable answers, and as they become applicable to more and more parts of a complex system, the need for those answers grows as well.

VII.Realize that maintenance is not free.

Since much of the motivation for the movement toward COTS-based systems is the expectation of easier and cheaper system maintenance, we now examine some aspects of how maintenance and upgrade activities can be affected in a system with numerous commercial components.

First, upgrading a COTS-based software system means that as new releases of the commercial components are made by the various vendors, the system will incorporate them. Though it is possible to occasionally skip over a release (e.g., ignore a release of something like "3.03A"), it is still the case that vendors tend to support only a limited number of versions, and ignoring a vendor's new releases cannot survive in the long term. It is also preferable that a system's commercial components should be as up to date as possible. A system with several commercial components thus has a very heavy dependency on various release cycles of the COTS vendors.

A further complicating factor is that different pieces of the system will be upgraded at widely varying intervals; licenses will be invalidated and need to be revalidated for different parts of the system at random intervals. And it should be kept in mind that component upgrade can result in numerous unforeseen problems: incompatible files and databases; different naming conventions; introduction of new conflicts between COTS components; these things are not at all uncommon. Depending on the number of COTS components and different COTS vendors, the effect of these multiple dependencies can vary from short-term user inconvenience to total system instability.

Another costly item for maintenance of COTS-based software systems results from the degree to which those systems exhibit an integrated character. We earlier suggested that the realities of the software world are quite different from those of the hardware world, and COTS software components are seldom built to "plug" into an existing system easily. The usual way to overcome this deficiency and build integrated systems that involve heterogeneous COTS software components involves "wrappers," "bridges," or other "glueware." These terms refer to third-party software that performs whatever integrating functions are necessary: trapping output from one component and reformatting it for input to another, sending notification messages about one tool's completion to another for start-up, and so forth. Most heterogeneous software systems that exhibit "integratedness" are built in precisely this way.

However, this technique *is not* one that leads to lower maintenance costs. First, writing wrappers can itself be a complex activity, requiring expertise both at the detailed system level as well as in the COTS components being wrapped. Second, wrappers are often "point-to-point" solutions, which means that when a vendor releases a new version, any "wrapping" involving the new component will potentially need upgrading. Given the random vendor release cycles described above, then keeping the glueware current and up to date for an integrated system of any complexity can become a maintenance nightmare.

VIII.You are not absolved of the need to engineer the system well

The discipline of engineering is no less critical to a COTS-based system than any other type of system; in some particulars it could be even more critical. And the reality of today's available COTS products is that few of them are designed to work together. Many have been created to be used stand-alone, and to require no co-location (let alone interaction) with any other product or component. Even when they have been designed to cooperate with another product, it is most often another product from the same vendor or from another vendor with whom the first vendor shares some special interest.

A COTS-based system is still a system with its own requirements, both developmental and lifecycle. Although the parts might be obtained from commercial sources, no one cares about *the system itself* except the person who will pay for it, maintain it, and use it. This system will need to be designed, brought together, tested, and managed just the same as any other system you have built or acquired in the past. There are no magic formulae for this. Nor is the Government's responsibility for its systems eliminated by the new-found reliance on COTS products.

IX. Just "doing COTS" is not an automatic cost-saver

As of yet, there is little real Government experience with building and fielding significant COTS-based, especially over any significant part of an overall system lifecycle. So we have far more promises and wishful thinking than facts or verified cost models on which to base our enthusiasm for the use of COTS products in Government systems.

And what is behind this enthusiasm? At least in part, it is the observation of COTS-based revolutions in other industries (e.g., automobiles). These appear to hold out the hope and expectation that the use of COTS products will similarly lower software system costs. And the analogy is compelling, since it makes sense that the proration of development and upgrade costs across multiple users and competition between suppliers should lead to cost savings for each part of any system that derives from the COTS world.

But note that most of this argument is based on per-unit component cost considerations. What about the *system c*osts? For instance, consider the effort to understand the COTS products as a (potential) system component - a glossy brochure and vendor promises are not enough. And sometimes all the available documentation together is not enough to understand the full behavior of a product nor the implications of that behavior for the integration of the component into the system. Some other hidden costs include:

- market research to find the COTS products that are suitable
- product analyses to select among alternatives
- licenses and warranties, especially if the warranty available to the general public does not suit your needs

In addition, many of the aspects of COTS-based systems previously cited in this paper can have still greater costs:

- integration of all those diverse products into a system
- keeping up with the asynchronous upgrades of the various products (this includes re-acquiring the knowledge needed to be sure each will continue to be a contributing member of the system, not to mention individual upgrade costs -- vendors don't hand out new releases for free!)

- coordination of the host of support vendors during the system lifecycle
- recovery when a vendor discontinues a product or goes out of business altogether

In any given situation, which costs will prevail? The answer is not an automatic result of "doing COTS," but instead depends on the individual system and circumstance, the specific risk mitigation strategies, and the management skills at hand.

X. Just "doing COTS" must be part of a large-scale paradigm shift

In most college curricula today, the introduction to software and computer science still consists of learning one or more programming languages. This teaches people to write whole systems and subsystems, designing them from a blank piece of paper, then coding, debugging, and testing them.

In contrast, the use of existing products as components in a system requires determination of how to get them to cooperate with one another to achieve the goals of the system. This will often result in writing wrappers to achieve the desired cooperation and integration. And this will almost certainly eventuate in repeating these two steps again during maintenance as each product changes (usually independently) to keep the set of components continuously cooperating

These two paradigms are very different, and the move to the generation of COTS-based systems constitutes a significant paradigm shift for programmers and system developers. Extrapolating from that, we find that it also constitutes a significant paradigm shift for the testing, quality assurance, and maintenance personnel as well. And changes to all these positions require changes and paradigm shifts in managers, in the expectations they have and in the techniques they employ.

In other words, the change to COTS-based systems is not just a technological change. It affects many people in many roles in profound ways. Further, such changes for individuals are only the start of the change. Organizations can be equally impacted, experiencing changes in the activities they undertake, their structure and their relationships, required training, the corporate policies, the relationships between government and contractors, and relationships across the marketplace.

This paradigm shift toward integration of others' products, from a producer to a consumer mentality, has widespread effects. The worst thing one can do is to treat it as merely as a change in technology.

3. Afterthought

Much has been made, and will continue to be made, of "the software crisis." It periodically reasserts a presence in the Government's consciousness, and its symptoms -- spiraling software costs, growing system complexity -- are indeed valid causes of concern. But perhaps this phenomenon is less a "crisis" than is generally thought. Perhaps it is simply the case that the growth of software's cost and complexity is (at least partially) the natural result of the engineering decisions being made. That is, given that a greater and greater proportion of a system's functionality is being allocated to software, while at the same time the system's functional capabilities are themselves increasing, then it is inescapable that the cost and complexity of the software will proportionately

grow as well. This is the natural result of a massive reallocation of system behavior from one area to another. That this reallocation engenders a “crisis” is too often caused less by the software itself than by key *system* decisions being made without proper understanding of the constraints and limitations of software, or without appropriate input from software (as opposed to system or hardware) experts.

There is another factor that relates to this impending “crisis,” namely, the belief that more complex systems can be had for less cost. While it is sometimes the case, certainly with computer hardware, that capability grows as cost diminishes, it is doubtful that this is always a reasonable expectation. In the case of software, it flies in the face of numerous bitter experiences, and in the face of simple folk wisdom as well (“you get what you pay for”). Thus, while such slogans as “better, cheaper, faster” represent attractive ideals, it is naive in the extreme to believe that simply stating the slogan will make it come true. A more realistic goal should be to demand awareness, on the part of all high- and middle-level personnel, of the pragmatic realities of software-intensive systems. Thus, it is not reasonable to speak of buying “fully integrated CASE environments” when there is no agreement among environment experts even as to what the phrase “fully integrated” actually means. It is not reasonable to expect “completely interoperating software systems” to appear on demand when in spite of recent advances in gluing disparate systems together, it is still the case that Apple users and Sun users find themselves at an impasse when trying to share a word processor file. And it is not reasonable for the Government to continue the cycle of large-scale procurements, initiatives, thrusts, plans, or whatever else (most of which have consumed many precious dollars and provided precious little benefit) without a more solid grounding in the technical realities of software practice.

This grounding must be based on knowledge, not slogans. It is all well and good to hope that using commercial software components will save money in the long run: as we stated on the first page of this paper, we fully concur with the ideal of using COTS products when appropriate and when warranted. But the savings will not come automatically, and they will not be the result of applying a simplistic solution to a complex problem. Anyone who promises otherwise is offering us a Golden Calf to worship. Following that path didn’t work in years gone by. It is not likely that it will work today.